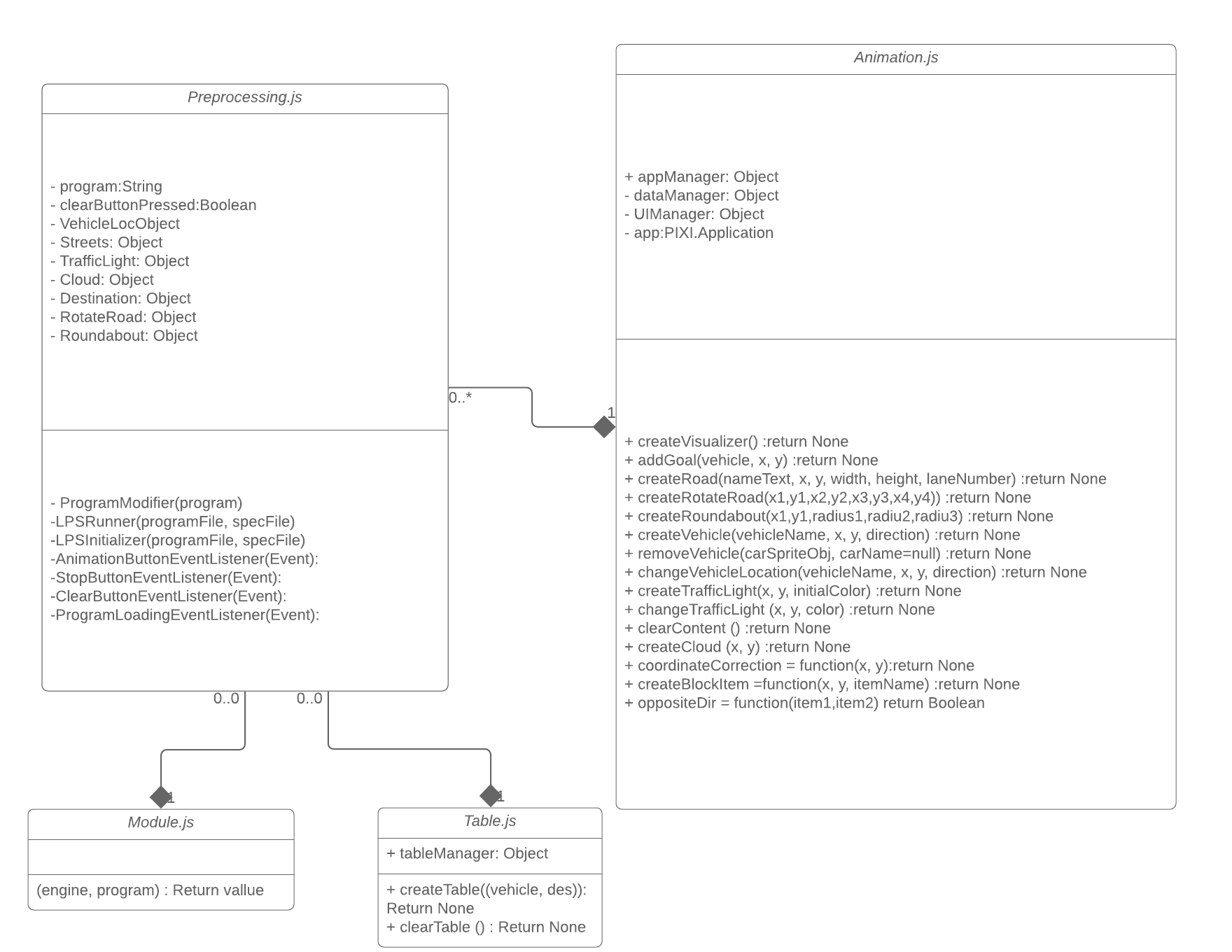
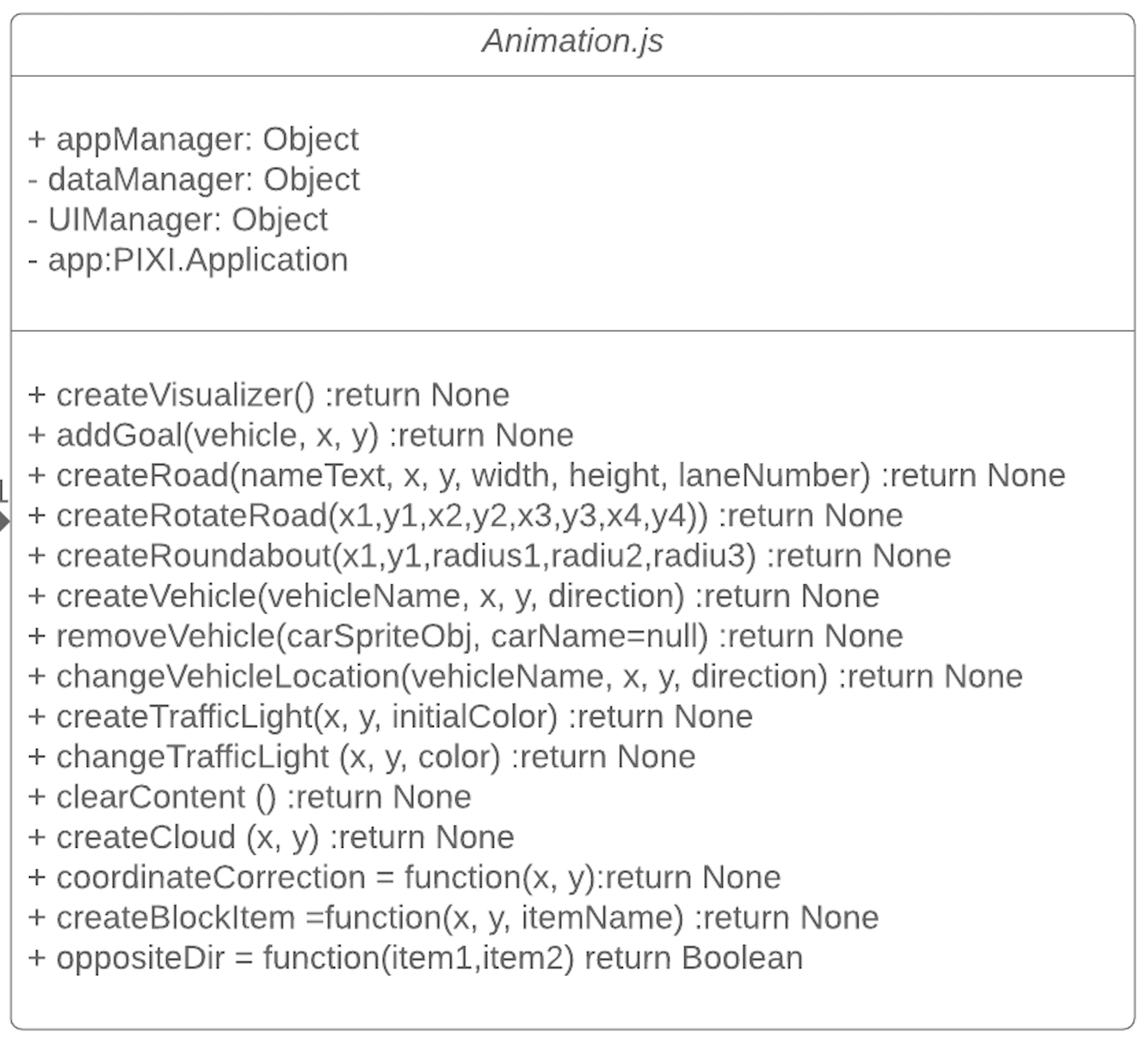
Front-end logic design and animation

This chapter I will introduce the underlying front-end side of LPS visualiser. Which include three main part. Part one is the animation part which is mainly about the PIXI canvas. The second part is the link between the animation and data (fluent/facts) how is the data parsed. The third part is about the layout design using twitter bootstrap and other open source toolkit.



The class diagram of the front-end side is structured as above. The

3.4.1 Animation.js

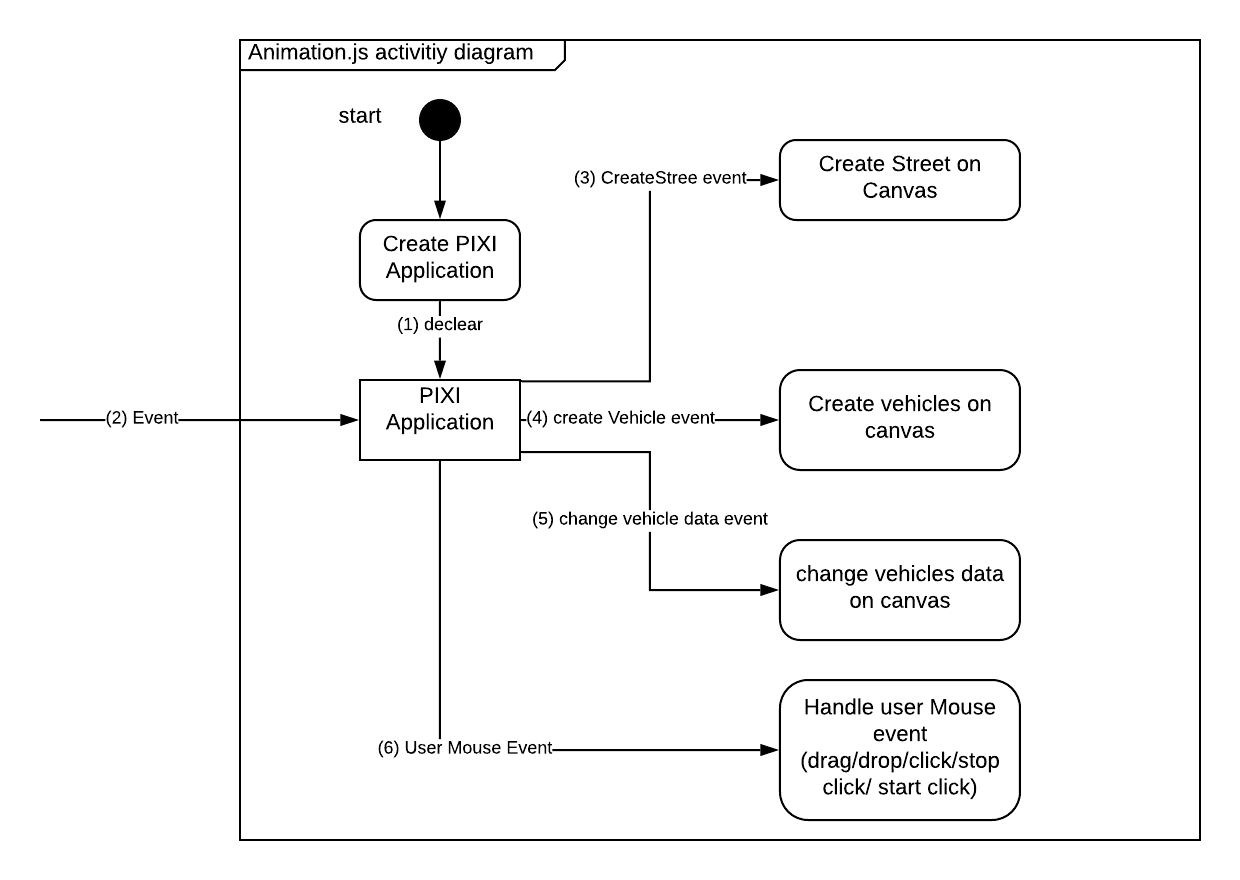


3.4.1.1 introduction

The Animation.js create the PIXI canvas application for animation and read the changing fluent in LPS program simultaneously. To achieve this, a concurrency model need to be used. Animation.js use a range of data structure such as AppManager to matain different list of object, UIManager to maintain a range of UI component.

3.4.1.2 structure

The structure of the animation is showed as an activity flow chart. The animation.js will run once the user is on the animation page. Due to the DOM structure introduced in Section? The PIXI application will be created as soon as the page is loaded. The different evnet happeded will fire different function. (1) is the creation of the PIXI applition. PIXI has a tree data structure every component play a child node role in the whole application whenever a component is created it need to be append into the application tree.



1. Describe about the creation of the PIXI application
2. Is different event whether is the event of the changing state of fluent or the user mouse event
3. Handles the street creation. When street predication as a fact in LPS program is read into the LPS interpreter.
4. Handles the creation of the vehivle When vehicle predication as a fluent in LPS program is read into the LPS interpreter.
5. Handles the information changing of the vehivle, including direction, location, speed while the runtime of lps program. State of location predication will be detected by the engine and the PIXI applition handles it by change the location on canvas accordingly.
6. Handles the user event while includes the clicking event (click on car to change direction, click the stop button, click the animation button). Drag event, drop event(drop in deletion, drop on street). Some globel information might change depends on the action that user perform.

3.4.1.3 class fields

There are four key components for Animation.js

/\*  
\* appManager is the object that allow other file to access the animation.  
\* \*/  
var appManager = {  
 vehicle: [],  
 street: [],  
 lights: [],  
 block: []  
};  
  
/\*  
\* data manager consist of the global information of the number of car and the response using to detect user click event  
\* \*/  
var dataManager = {  
 responseTime : 0,  
 carCounter:0  
  
};  
  
/\*  
\*UI manager manage the UI field including the graphics, button and textures.  
\* \*/  
var UIManager = {  
 graphics : null,  
 richTextTitle: null,  
 richTextAction: null,  
 xAxisText:null,  
 yAxisText:null,  
 originPointText:null,  
 textureButtonPlus : null,  
 textureButtonPlusOver : null,  
 textureButtonPlusDown : null,  
 textureButtonMinus : null,  
 textureButtonMinusOver : null,  
 textureButtonMinusDown : null,  
  
 buttonPlus : null,  
 buttonMinus : null  
};  
  
/\*  
\*The main application framework to host the whole canvas with height etc are defined here  
\* \*/  
var app = new PIXI.Application({backgroundColor: 0x1099bb, width: 1300, height: 1000});

Firstly, the appManager maintain the fluent entered by user such as vehicle, street, light, blocked item and so on.

DataManager maintain the response used to detect user whether drop and drop the vehicle or just click on the vehicle to change the direction. Car counter is used here as an Integer type also used here

3.4.1.4 functionality

createVisualizer Initialise the canvas, all the buttons, title text, define the base graphic

addGoal = function(vehicle, x, y) adding goal tag onto the map

createRotateRoad = function(x1,y1,x2,y2,x3,y3,x4,y4) This field will create the rotated road depends on the user defined

reateRoundabout = function(x1,y1,radius1,radiu2,radiu3) This field will create the roundabout

createVehicle = function (vehicleName, x, y, direction) This field will create the vehicles with name, location and direction.

removeVehicle = function(carSpriteObj, carName=null) Will remove the vehicle if user drop the vehicel in the deletion area

changeVehicleLocation = function (vehicleName, x, y, direction) this field will modify the child in app

createTrafficLight = function(x, y, initialColor) this field will create the traffic light based on the location

changeTrafficLight = function(x, y, color) this field will change the traffic

clearContent = function ()this method will clear out the street and vehicle information

\* the basic canvas will remain the same (coordinate axis and the title text)

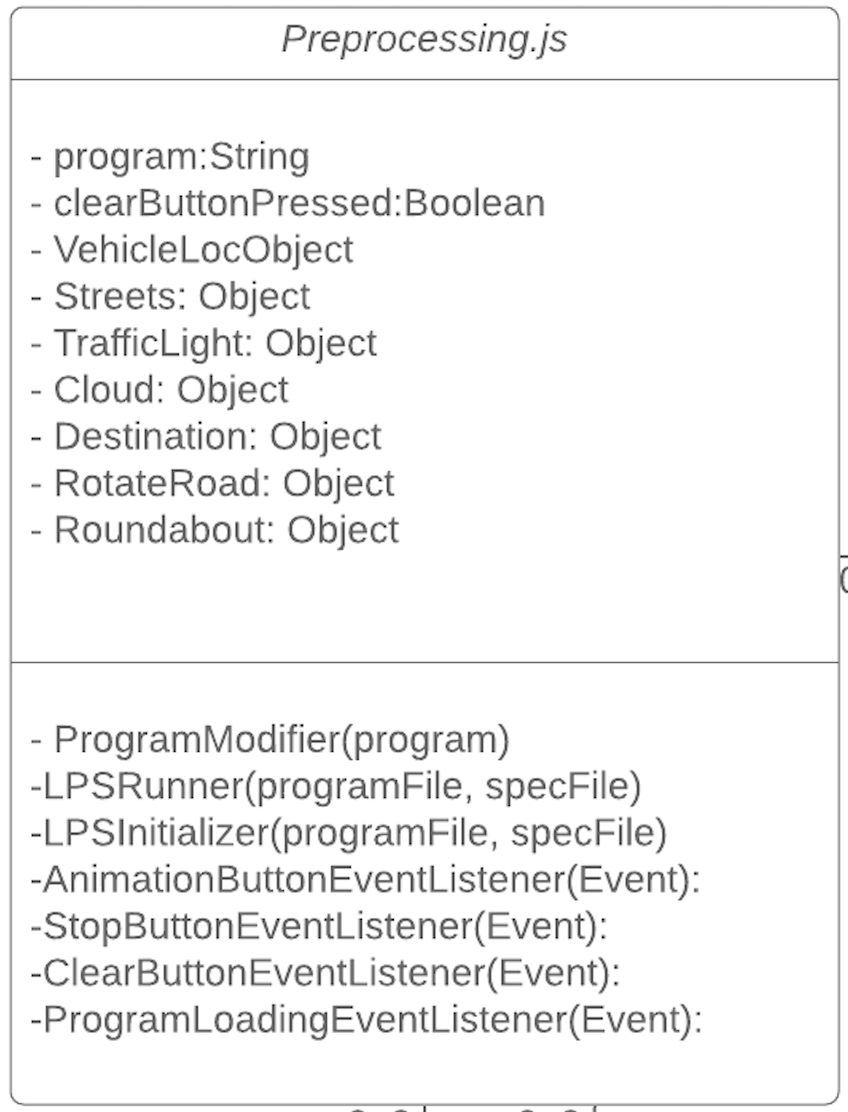
createCloud = function(x, y) this method will create the cloud and animate the cloud in the canvas

coordinateCorrection = function(x, y) correct the coordinate on to the street. use might place the car onto other part rather than the road

createBlockItem =function(x, y, itemName) When there is a blocked item placed by user/program the front end will show up a cross

oppositeDir = function(item1,item2) This part is checking all the adding cars that is there is any car that is opposite to each other (same horizontal or vertical level)

3.4.2 preprocessing.js



3.4.2.1 introduction

3.4.2.2 structure

3.4.2.3 Document Object Model (DOM)

3.4.2.4 functionality

3.4.3 Demonstration Website design

3.4.3.1 structure

3.4.3.2 API and open source toolkit

3.5 back-end hosting

3.5.1 back-end hosting

3.5.1.1 introduction

3.5.1.2 structure

3.5.1.3 Express.js

3.5.1.4 MongoDB and mongoose

3.5.1.5 Passport.js

3.5.1.6 AWS EC2 server